Introduction to Embedded Systems – Part I

STUDY MATERIAL

Answers to Objective Questions

Introduction to Embedded Systems – Part I

CHAPTER 1

1. b) Special Purpose
2. d) a) or c)
   Embedded Systems are Electronic/Electro mechanical systems which are designed to perform a specific function and are a combination of both hardware and firmware (Software).
3. b) Always contain an operating System
   It is not mandatory that all embedded systems contain an operating system for its functioning
4. c) Cell phone
5. b) Apollo Guidance Computer (AGC)
6. c) Autonetics D-17
7. d) All of these
   Embedded Systems are designed to serve the purpose of a single function like data collection, data processing, data communication monitoring and control or a combination of them
8. b) Network Router
9. b) Monitoring
10. a) Apple iPOD (Media Player Device)

CHAPTER 2

1. d) Any/All of these
2. a) Store the lower-order byte of the data at the lowest address and the higher-order byte of the data at the highest address of memory
   Little-endian processors store the lower-order byte of the data in memory at the lowest address, and the higher-order byte at the highest address.
3. b) 0x00
   Since the integer is represented as 4 bytes by the system, the higher 3 bytes will be 0 (0x00) and the lower byte is 255 (0xFF) and it is represented as 0x000000FF. The Big endian architecture stores the higher order data byte in memory at the lowest address. This is represented as

4. a) Simple and lesser in number

STUDY MATERIAL

Introduction to Embedded Systems – Part I

5. e) All of these
6. a) Harvard
   Microprocessors/controllers based on the Harvard Architecture will have separate data bus and instruction bus. This allows the data transfer and program fetching to occur simultaneously on both the buses. With Harvard architecture, the data memory can be read and written while the program memory is being accessed. These separated data memory and code memory buses allow one instruction to execute while the next instruction is fetched and there by supports easier instruction pipelining.
7. a) True
8. b) PROM
9. e) a) or b)
   EEPROM and FLASH are erasable and supports re-programming and are best suitable for development work
10. a) True
11. a) True
12. b) False
    Execution of program from RAM is faster than the execution from ROM
13. b) False
    Dynamic RAM stores data in the form of charge
14. a) True
15. e) b) and c)
    Wi-Fi and Bluetooth are examples for wireless communication interface
16. f) both b) and d)
    Atmel Automotive AVR and Microchip CAN PIC microcontrollers are exclusively designed for automotive applications and they integrate automotive application specific peripherals and communication units.
17. b) 8192
    1Kb = 1024 Bytes. 1 Byte memory = 8 memory cells. Hence 1Kb memory contains 1024 x 8 = 8192 memory cells.
18. b) NOR FLASH
19. a) 1024
    The size of Serial EEPROM is specified in terms of Kilobits. Hence 1 Kb serial EEPROM memory contains 1024 memory cells.
20. e) only a) and b)
    Digital to Analog Converter (DAC) is normally used as an output subsystem
21. d) All of them
    Optocoupler can be used in both input and output subsystem.
22. c) Optocoupler can be used in both input and output circuitry
23. b) Contains two windings per stator phase
24. d) d) Both a) and b)
25. b) 8
    16 keys can be arranged in a 4 column x 4 row matrix. Hence 8 I/O lines are required
26. a) 6x4
27. e) Only a) and c)

Introduction to Embedded Systems – Part I

Bluetooth is normally used as a communication interface for communicating with external device and not for communicating between ICs within the board.

28. b) 2
    I2C requires a minimum of two wires namely Serial Data and Serial Clock for implementing data communication
29. c) 4
    SPI requires a minimum of 4 lines namely Master In Slave Out (MISO), Master Out Slave In (MOSI), Clock, and Device select for implementing data communication
30. e) Only a) and b)
    I2C and SPI are synchronous serial interfaces. They require a clock for data communication. UART is an asynchronous serial communication interface. UART devices communicate under a pre-defined set of communication rules (like baudrate, start bit, stop bit etc) and doesn't require a clock for data communication.
31. b) False
    RS232 is an extension of the UART serial interface and it doesn't require a clock for data communication.
32. c) 127
33. a) ZigBee Coordinator
34. d) 171.2Kbps
35. d) 8

CHAPTER 3

1. a) True
2. d) All of these
3. c) SCADA system
4. b) Non functional requirements
5. c) Both
   Response is a measure of quickness of the system. It gives an indication about how fast your system is tracking the changes in input variables
6. c) Both
   Throughput deals with the efficiency of a system. In general it can be defined as the rate of production or operation of a defined process over a stated period of time
7. c) a) or b)
   A 'Benchmarks' is a reference point by which something can be measured. Benchmark can be a set of performance criteria that a product is expected to meet or a standard product that can be used to compare other products of the same product line.
8. a) True
   Reliability is a measure of how much % you can rely upon the proper functioning of the system or what is the % susceptibility of the system to

Introduction to Embedded Systems – Part I

failures. Mean Time between Failures (MTBF) and Mean Time To Repair (MTTR) are the terms used in defining system reliability.

9. a) True
10. c) Both of these

    A highly reliable system is less susceptible to failures and there by the corrective maintenance requirement for such a system is very less. Also the availability of highly reliable system is very high.

11. d) All of these

    If the Mean Time Between Failure (MTBF) for a product is high, the reliability and availability of the product is high. Also the chances for product breakdown and the requirement for corrective maintenance will be low.

12. c) 89%

    The availability of an embedded system is expressed as
    Availability = MTBF/ (MTBF+MTTR)
    The MTBF for the given system is 4 months
    The MTTR for the given system is 2 Weeks = 0.5 months
    Availability = (4/ (4+0.5)) * 100% = (4/4.5) * 100% = 88.88% ~= 89%

13. b) Confidentiality, Integrity, Availability

    Confidentiality, integrity and availability are the three major measures of information security.

14. b) Confidentiality
15. b) UV emission from the embedded product

    'gradual' safety threats are safety problems that develop over a period of operation of a product. Hazardous emission from a product is a typical example for this.

16. d) a) and c)

    The non-functional requirements of a product 'not' on the basis of operational aspects are known as Non-operational quality attributes.

17. b) Safety
18. d) Portability
19. a) True

    In the Information security context, Confidentiality deals with the protection of data and application from unauthorized disclosure.

20. a) Hardware & Firmware debug-ability
21. c) Both of these

    For an Embedded System, the quality attribute Evolvability refers to the ease with which the embedded product (including the firmware and hardware) can be modified to take advantage of new firmware or hardware technologies. It can be an upgrade of the existing product or a modification of the existing product

22. a) True

    Portability is a measure of 'system independence'

23. a) Product launching
24. b) Product maturity

Introduction to Embedded Systems – Part I

CHAPTER 4

1. e) a) and b)
2. d) Mirror Control
3. b) Volkswagen 1600
4. a) True
5. d) All of these
6. d) All of these
7. d) All of these
8. e) All of these

CHAPTER 5

1. b) 128 bytes
2. b) 128 bytes
3. d) 4 Kbytes
4. c) 32
5. b) Four 8 bit bi-directional ports
6. a) Two 16 bit Timers
7. c) Full duplex with configurable baudrate
8. a) Harvard Architecture
9. d) All of these
10. b) 16bit wide
11. c) Port 0 acts as the multiplexed address/data bus and Port 2 acts as the higher order address bus
12. b) PC
13. a) DPTR
14. a) 8bit wide
15. d) 4
    With 256 bytes/page, it requires 4096/256 = 16 Pages for representing 4K bytes (4096 Bytes). For representing 16 pages x port lines is required. x is calculated as $2^x = 16$
    x = 4
16. e) a) b) and d)
    The Von-Neumann architecture shares a common data and code memory space. Hence the program memory and code memory should be combined to generate Von-Neumann memory model. The code memory read signal (PSEN\) is available externally when the code memory is external to the controller. The data memory read is controlled by the signal RD\. Both RD\ and PSEN\ are active low. RD\ and PSEN\ are ANDed to generate the Output Enable pulse (OE\) for the memory chip holding the instructions and data. The controller is informed that the code memory is external to the chip through the line External Access (EA\). EA is an active low signal and pulling EA to logic low makes the program execution happen from the external memory chip.
17. b) 8

Introduction to Embedded Systems – Part I

Registers R0 to R7 (Memory locations 00H to 07H) are the general purpose registers implemented in standard 8051 architecture. The Registers R0 to R7 are available in 4 register banks. But only one bank is active at a time.

18. e) 128
19. a) 0
20. c) F0H
21. d) Random data
    Accessing the upper 128 memory through indirect addressing returns the scratchpad memory data if it is physically implemented. If the Upper 128 bytes of scratchpad RAM is not physically implemented, reading from that location returns random data
22. c) PSW
23. c) 2
    f1 (PSW.1) and f2 (PSW.5) are the two general purpose flag available in the Program Status Word (PSW) for programmers.
24. a) 1
    The Parity flag 'P' present in the PSW is set when the number of 1's present in the accumulator data is odd. Otherwise it is reset.
25. b) OV = 0; C = 0; AC = 1
    For addition operation, the Overflow Flag (OV) is set when there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared, the Carry flag C is set when a carry is generated out of bit 7 of accumulator, the Auxiliary Carry flag (AC) is set when a carry is generated out of bit 3 of accumulator. For subtraction operation, the carry (borrow) flag is set if a borrow is needed for bit 7, and clear C otherwise. (If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand.)AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.
26. d) 10H
    RS0 = 0 and RS1 = 1, represents bank 2 for general purpose registers (R0 to R76) and the corresponding physical address for the general purpose registers lies in the range 10H to 17H (R0 at 10H and R7 at 17H).
27. a) 2T States
28. a) Port 0
29. b) Logic 1
30. c) A5H
    During the execution of MOVX@DPTR instruction, the Port 2 latch does not have to contain 1s, and the contents of the Port 2 SFR are not modified.
31. a) 1
    If P3.x bit latch contains a 1, then the output level is controlled by the signal “alternate output function”. The actual P3.x pin level is always available to the pin’s alternate input function, if any.
32. e) a) or b) or c)
33. b) External 0

Introduction to Embedded Systems – Part I

For equal priority interrupts, the priority resolution mechanism adopted is internal polling sequence. External 0 interrupt is polled first and it is always assigned highest internal priority.

34. c) 8 Byte
35. c) 3 Machine Cycle
36. b) 5 Machine cycles
    MUL AB is a 4 machine cycle instruction and the external 0 interrupt is asserted and latched at S5P2 of the first machine cycle of *MUL AB* instruction. Hence it requires 3 more machine cycles to complete the execution of the instruction in progress. Another 2 machine cycle is required for generating an LCALL to the ISR location of the interrupt.
37. a) 1Machine Cycle
38. b) Oscillator Frequency/12
    In the "Timer" function, the timer register is incremented in every machine cycle. Since one machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.
39. d) Oscillator Frequency/24
    In the "Counter" function, the timer register is incremented in response to a l-to-0 transition at its corresponding external input pin, T0, T1. In this mode, the external input is sampled during S5P2 of every machine cycle. When the samples show high in one cycle and a low in the next cycle, the count is incremented. The new count value appeared in the timer register during S3P1of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 oscillator periods) to recognize a l-to-0 transition, the maximum count rate is $1/24^{th}$ of the oscillator frequency.
40. b) Timer 1
41. c) Mode 2
42. a) 8bits of TH0/TH1 and least significant 5 bits of TL0/TL1
43. d) a) and c)
    The serial port of standard 8051 is full duplex and receive buffered.
44. c) SBUF
45. b) The 'Baudrate' is given as Oscillator Frequency/12
46. c) 9600
47. b) 0000H
48. e) c) or d)
49. e) All of these
    In the '*Idle mode*', the internal clock signal is gated off to the CPU, but not to the Interrupt, Timer, and Serial Port functions. The Stack Pointer, Program Counter, Program Statue Word, Accumulator, and all other registers retain their data during Idle mode. The port pins hold the logical states they had at the time Idle was activated. ALE and PSEN hold at logic high levels.
50. d) a) and b) only

Introduction to Embedded Systems – Part I

CHAPTER 6

1. f) All of the above
2. c) Immediate
   The '#' in the operand list represents an immediate constant
3. a) Direct
4. b) Indirect
   The '@R0' or '@R1' in the operand list represents indirect memory addressing
5. d) None of the above
   The '@A+DPTR' in the operand list represents indexed memory addressing. It is used for accessing the code memory
6. c) 22H
   The instruction *MOVC A, @A+PC* is one byte and it is located at code memory address 004EH. Code memory 004FH contains the machine code (22H) corresponding to *RET*. On executing the *MOVC A, @A+PC*, the PC will be incremented by 1 (becomes 004FH). With A as 00H, A+PC becomes 004F and the *MOVC* instruction loads accumulator with the contents of code memory location 004FH, which is 22H
7. d) R0 = 50H; A = 50H
   The instruction *MOV <destination>, <source>* copies the content of <source> to <destination>. The source remains unchanged. Here Accumulator is the destination and R0 is the source
8. c) Data memory location 00H = F0H; SP = 08H
   The *PUSH <location>* instruction increments the Stack Pointer register SP by 1 and saves the content of memory location pointed by <location> at the memory location pointed by SP register. The contents of memory location <location> remains unchanged
9. c) Memory location 00H = 0FH; Memory location 08H = 0FH; SP = 07H
   The *POP <location>* instruction loads the memory location <location> with contents of the memory location pointed by the SP register. The SP register is decremented by 1 after the content is retrieved. The contents of memory location pointed by SP register remains unchanged
10. c) Memory location 0FH = FFH; Accumulator = 00H
    The *XCH A, <location>* instruction interchanges the content of accumulator and memory location pointed by <location>
11. c) Memory location 0FH = AAH; Accumulator = 55H; R0 = 0FH
    The *XCHD A, @Ri* instruction interchanges the low nibble (lowest 4 bits) of accumulator with the low nibble of the content of the memory location pointed by Ri (i = 0 or 1)
12. c) P2 SFR = 20H; P1 SFR = 50H during the first machine cycle; WR\ signal is asserted once
    The *MOVX @DPTR, A* instruction writes the content of accumulator to the external memory address pointed by DPTR register
13. a) The program memory is external to the controller
14. b) 2
15. b) 2

STUDY MATERIAL

Introduction to Embedded Systems – Part I

16. b) 2
17. a) 1
18. c) Both a & b
    The *MOVC* instruction uses indexed addressing mode and it is used for reading from the program memory
19. d) Accumulator = 00H; Carry Flag = 1
    The *ADD* instruction sets or clears the carry flag according to the result of ADD operation. If an overflow (result >FFH) occurs as a result of ADD operation, the carry flag is set. Otherwise it is cleared
20. b) 0
    Refer the explanation given for the previous question
21. d) Accumulator = 00H; Carry Flag = 1
22. d) Accumulator = 1FH; Carry Flag = 1
    If the result of subtraction operation is -ve (i.e. Accumulator content < the value to be subtracted), the borrow flag is set and the accumulator content will become the 2's complement of the difference between the two numbers. Here the difference between the two numbers is E1H (F0H-0FH). The 2's complement of E1H is 1FH
23. b) Accumulator = E1H; Carry Flag = 0
    Refer the explanation given for the previous question
24. a) Accumulator = 0FEH; B = 01H
    The *MUL AB* instruction multiplies the accumulator content with B register content and stores the lower order byte of the result in accumulator and the higher order byte of result in B register. Here accumulator contains 0FFH and B holds 02H. Multiplying Accumulator with B register results in 01FEH. Accumulator holds the lower order byte of the result, FEH and B holds the higher order byte of the result 01H
25. d) Carry Flag = 0; Overflow Flag = 1
    The *MUL AB* instruction does not modify the carry flag. If the result of *MUL AB* instruction generates an overflow (Result is greater than 0FFH), the overflow flag is set, and otherwise the overflow flag is cleared.
26. b) Carry Flag = Remanis same as the previous value; Overflow Flag = 1
    Refer the explanation given for the previous question
27. b) Accumulator = 7FH; B = 01H
    The *DIV AB* instruction divides the accumulator content with B register content and stores the integer part of result in accumulator and the remainder in B register. Here accumulator contains 0FFH and B holds 02H. Dividing 'Accumulator' with B register results in 7FH with remainder 01H. Accumulator holds the result, 7FH and B holds the remainder 01H
28. c) Accumulator = Undefined; B = Undefined; Overflow Flag = 1
    The divide by zero condition results in undefined result. The accumulator and B register contents are undefined. The divide by zero condition is indicated by setting the overflow flag
29. b) Accumulator = 00H; Carry Flag = 0
    The *INC A* instruction increments the content of Accumulator. The accumulator content simply rolls over to 00H if its value is FFH before

executing the *INC* instruction. The carry flag remains unchanged on executing the *INC* instruction.

30. c) Accumulator = FFH; Carry Flag = 0
    The *DEC A* instruction decrements the content of Accumulator. The accumulator content simply rolls over to FFH if its value is 00H before executing the *DEC* instruction. The carry flag remains unchanged on executing the *DEC* instruction.
31. d) 204FH
    The set of instructions decrement the content of DPTR register
32. b) 15 H
    The *DA A* instruction adjusts the accumulator for decimal. If the Accumulator nibbles are greater than 9H (AH to FH), 6 is added to it to bring it to a valid decimal number representation.
33. b) 3AH
    The 8051 doesn't have a way to differentiate an addition as BCD addition. It treats all addition operation as binary addition. However it provides an instruction DA A for adjusting the accumulator content to a valid BCD number, if the numbers added are binary numbers. The *DAA* instruction should follow the *ADD* instruction. Otherwise the result of addition will be an invalid BCD number.
34. a) 40H
    Refer to the explanation given to the previous question.
35. c) FFH
    The *ORL A ,#0F0H* instruction performs a bitwise ORing of accumulator content (0FH) with F0H. This will result in FFH and the result is stored in Accumulator
36. d) 00H
    The *ANL A, #0F0H* instruction performs a bitwise ANDing of accumulator content (0FH) with F0H. This will result in 00H and the result is stored in Accumulator
37. b) 7FH
    The *XRL A, #0D5H* instruction performs a bitwise XORing of accumulator content (0AAH) with 0D5H. This will result in 7FH and the result is stored in Accumulator
38. d) Accumulator = 00H; Carry Flag = 1
    The *CLR A* instruction clears the accumulator (Load 00H to Accumulator). The Carry flag remains unaffected.
39. d) The LS Bit of the data held by data memory location 20H becomes 0
    The *CLR 00H* instruction clears the bit with address 00H. The bit addressable memory starts at data memory location 20H. Bit 00H represents the least significant bit of the content held by data memory location 20H
40. d) Accumulator = F0H; Carry Flag = 1
    The *CPL A* instruction complements the accumulator content. The carry flag remains unchanged.
41. b) Accumulator = 80H; Carry Flag = 1

Introduction to Embedded Systems – Part I

The *RL A* instruction rotates the accumulator content to the left by one bit. The MS bit is shifted out and moves to the LS bit position. The LS bit moves to the left.

42. c) Accumulator = 03H; Carry Flag = 0
    The *RLC A* instruction rotates the accumulator content to the left by one bit. The MS bit is shifted out to the Carry bit and the content of carry flag moves to the LS bit position. The LS bit moves to the left.
43. d) Accumulator = 80H; Carry Flag = 1
    The *RR A* instruction rotates the accumulator content to the right by one bit. The LS bit is shifted out and moves to the MS bit position. The MS bit moves to the right.
44. b) Accumulator = 80H; Carry Flag = 1
    The *RRC A* instruction rotates the accumulator content to the right by one bit. The LS bit is shifted out to the Carry bit and the content of carry flag moves to the MS bit position. The MS bit moves to the right.
45. b) F5H
    The SWAP A instruction swaps the lower and higher nibbles of accumulator
46. d) The LS Bit of the data held by data memory location 20H is complemented
    The *CPL 00H* instruction complements the bit with address 00H. The bit addressable memory starts at data memory location 20H. Bit 00H represents the least significant bit of the content held by data memory location 20H
47. c) Carry Flag = 1; P1.0 = 0
    The instruction *ANL C, /P1.0* logical AND the complemented value of P1.0 with Carry flag and stores the result in carry flag. The actual value of P1.0 remains unchanged
48. c) Carry Flag = 1; P1.0 = 0
    The instruction *ORL C, /P1.0* logical OR the complemented value of P1.0 with Carry flag and stores the result in carry flag. The actual value of P1.0 remains unchanged
49. a) AJMP
    The AJMP instruction encodes the destination address as 11 bit constant. The instruction is two byte long consisting of the opcode, which itself contains higher 3 bits of the 11 bit constant. Rest 8 bits of the destination address is held by the second byte of the instruction. When an AJMP instruction is executed, these 11 bits are substituted for the lower 11 bits in the Program Counter (PC). The higher 5 bits of the Program Counter remains the same. Hence the destination will be within the same 2K block of the current instruction. In general AJMP is used for jumping within a memory block
50. c) A1H, 0FH
    The AJMP instruction encodes the destination address as 11 bit constant. The instruction is two byte long consisting of the opcode, which itself contains higher 3 bits of the 11 bit constant. Rest 8 bits of the destination address is held by the second byte of the instruction. The Opcode for AJMP is 01H, which is represented by the least 5 bits (00001) of the first byte. The most 3 digits of the first byte holds the MS byte of the jump location 5H (101). Hence the first byte of the instruction becomes 1010000001 (A1H). The

second byte of the instruction represents the least significant 8 bits of the jump location (0FH)

51. e) only b) and c
    The AJMP and LJMP instructions encode the absolute memory location to which the jump is required as operand where as the SJMP uses the offset of the jump location from current location as the jump parameter.
52. a) Relative offset method

CHAPTER 7

1. b) Data Flow Graph
2. b) Data Flow Graph
3. c) Datapath Architecture
4. b) MIMD
5. a) Finite State Machine (FSM)
6. d) Hierarchical/Concurrent Finite State Machine Model
7. d) Communicating Process Model
8. e) b) and c)
9. a) Class
10. c) State Machine
11. c) Use case Diagram
12. b) Object Diagram
13. b) Use case Diagram
14. b) Sequence Diagram
15. a) Collaboration Diagram
16. d) All of these

CHAPTER 8

1. d) Zener Diode
2. e) Only a) and c)
3. e) a) and c)
4. d) a) and b)
5. e) Voltage Follower
6. d) XOR
7. b) 74LS244
   IC 74LS244 is an example for octal buffer (8 buffers in a single IC) with tri-state outputs
8. c) Multiplexer
9. b) False
   The output of a combinational circuit depends only on the present state of the inputs and doesn't depend on the previous output state. Hence they don't require a memory element for latching the previous states.
10. b) False

Introduction to Embedded Systems – Part I

The output of a Half Adder circuit is given as Y = AB\ + A\B. It is obvious that the output depends only on the present inputs and it can be implemented using simple combinational circuits.

11. c) D
12. c) Undefined
    If both the Set input (S) and Reset input (R) are at logic high, the output of the SR flip-flop will be in undefined state. It is not logical to assert both the Set and Reset inputs of an SR flip-flop simultaneously
13. d) T
    The T Flip-flop is known as 'Toggle' flip-flop. With the Input set to logic '1', the output of the T flip-flop toggles with each clock input
14. a) 1
15. a) 1
16. b) 0
    With the Input set to logic '1', the output of the T flip-flop toggles with each clock input
17. c) LSI
18. d) VLSI
19. c) Mixed Signal
20. b) The different components involved in a hardware product and the interconnection among them
    '*Schematic*' represents the different components involved in a product and the interconnection between the components. Whereas '*Layout*' represents the physical arrangement of the various components present in a product and the interconnection among them.
21. c) Both of these
    '*Bill of Materials*' (BOM) gives the list of various components (in terms of component type and value) present in the schematic and their quantity.
22. d) All of these
    '*Netlist*' is the soft form representation of the different components present in the schematic and the hard wired connections required among the various components. 'Netlist' file is the output of schematic design and it is fed as input to 'Layout' tool for generating the 'Layout' of the PCB.
23. c) All of these
    The '*Layout*' file contains the information on different components present, their physical appearance (footprint), the component placement, inter connection among the components, number of layers for inter connection etc… for the PCB. It is considered as the soft form representation of the 'Blueprint' of the PCB.
24. e) All of these
    Footprints, Routes, Layers, Vias and Marking texts/graphics are the building blocks of '*Layout*'.
25. a) The 'Top view' of a component
26. b) Single Inline Package (SIP)
27. e) only b) and c)

Introduction to Embedded Systems – Part I

TSSOP and SOIC are Surface Mount Packages, whereas PDIP is a 'throughhole' package.

28. e) a) and d)
    The Quad Flat Package (QFP) is the package containing pins/pads on all 4 sides of the package. VQFP (Very thin Quad Flat Package) and PQFP (Plastic Quad Flat Package) are examples for QFP package.
29. b) Route/Trace
30. a) conductive drill hole
    The '*via*' interconnects the connections routed through different layers.
31. e) a) and b)
    Assembly notes are printed on Assembly Bottom (ASB) or Assembly Top (AST) layers.
32. d) a) and c)
    '*Gerber file*' contains the component PCB layout info, routing info, drill details etc in a universally accepted file exchange protocol format. Gerber file is a collection of artworks corresponding to each layer of the PCB. It acts as an information exchange mechanism between PCB designer and fabricator
33. a) Only a single layer is used for routing the connections between components
    For single layer PCB, the routing of connections are done on only one side of the PCB (Either Top or Bottom layer). It is not mandatory that a single side PCB contains component placement on only one side. If the PCB contains both 'Surface mount' and 'Throughhole' components, the Surface Mount' component is placed at the side on which the routing is done and the 'Throughhole' components are placed on the other side.
34. d) All of these
    A '*Flexible PCB*' is highly flexible compared to the normal PCB and it uses a flexible substrate like 'plastic' for PCB etching. It is commonly used for the fabrication of Antennas, membrane keypads etc...
35. d) All of these
    Photo engraving (Photo etching), PCB milling and PCB Printing are the three commonly used PCB fabrication techniques.
36. f) only a) and c)
    Photo Etching and PCB milling are '*subtractive*' process in which copper is removed from the substrate, whereas PCB printing is a non-subtractive process, where copper is added through printing technique.
37. e) b), c) and d)
    '*Solder Mask*' is a non-conductive protective layer for protecting the copper tracks and copper plated area from corrosion. The protective layer is formed by materials like plastic. It also prevents the 'wetting' of solder on soldering the component. 'Solder Mask' layer is built on the PCB by the PCB manufacturer at the time of fabricating the PCB
38. e) b), c) and d)
    'Conformal Coating' is a non-conductive protective layer for protecting the various components from atmosphere interaction. The protective layer is formed by dilute solution of silicon rubber or epoxy, or plastic. *'Conformal*

Coating' layer is build on the PCB after soldering all the components on the PCB, by spraying the solution on the board or by dipping the board in the solution.

CHAPTER 9

1. b) Machine Language
2. a) Machine Language
3. a) Pseudo-op
4. a) Assembler
5. b) The machine code corresponding to a processor which is different from the processor of the PC used for application development
6. c) 0x12ff84
   0x12ff7c + (Size of integer in bytes * 2) = 0x12ff7c + (4 * 2) = 0x12ff7c + 8 = 0x12ff84
7. a) 0x12ff7c
   ptr++ is a post increment operation. The pointer *ptr* is incremented after executing the *printf* instruction.
8. b) 0x12ff7d
   ++*ptr* is a pre increment operation. The pointer *ptr* is incremented first and the *printf* instruction is executed after that.
9. c) Compile error (cannot add two pointers)
   Addition of two pointers are illegal. Since pointers holds memory addresses, the addition of memory address may result in the overflow (the result is greater than 32 bits for a 32 bit platform).
10. a) 0x0b
    The *++*ptr1;* operation increments the content pointed by the memory address held by *ptr1* and *ptr1 retrieves the content held by the memory location pointed by pointer *ptr1*. Here *ptr1* holds the memory address *0x12ff7c* and the memory address *0x12ff7c* holds the data *10* (*0x0a*). The instruction *++*ptr1;* increments *0x0a* by 1
11. c) 0x12ff7c
    The instruction *++*ptr1;* increments the content pointed by the memory location held by *ptr1* and **ptr1* retrieves the content held by the memory location pointed by pointer *ptr1*. The memory location held by *ptr1* remains unchanged on executing the instruction *++*ptr1;* The instruction *printf("%x\n", ptr1);* prints the memory address held by *ptr1*
12. b) The contents of memory location 0x12ff7d
    The instruction **ptr1++;* increments the pointer and not the content pointed by it. Here the pointer *ptr1* holds the address *0x12ff7c*. Executing the instruction **ptr1++;* increments the pointer to *0x12ff7d*. The instruction *printf("%x\n", *ptr1);* prints the content held by memory location *0x12ff7d*.
13. d) 0x12ff7d
    The instruction **ptr1++;* increments the pointer and not the content pointed by it. Here the pointer *ptr1* holds the address *0x12ff7c*. Executing the

STUDY MATERIAL

Introduction to Embedded Systems – Part I

instruction **ptr1++;* increments the pointer to *0x12ff7d*. The instruction *printf("%x\n", ptr1);* prints the memory address held by the pointer *ptr1*.

14. c) \0
15. b)5
    The function *strlen* returns the number of characters present in the string excluding the string terminator character '\0'
16. c) Compile error
    Addition of two strings using the addition operator '+' is illegal.
17. b) 0
    The *stricmp* function is not case sensitive.
18. d) World!
    The *strcpy(str1,str2)* function copies string 2 (*str2*) to string 1 (*str1*). Here *str2* is *"World!"*
19. c) Compile error
    Comparison of two strings using the equal to (=) operator is illegal.
20. d) 0 bytes
    Defining a structure will not allocate memory for it.
21. f) Content of memory location 0x7001
22. b) 2
    The *offsetof* macro returns the offset, in bytes, of the *member* element from the beginning of struct *structure*. In this example the member variable *seconds* is placed at an offset of 2 bytes from the beginning of the structure *RTC_Time*
23. b) 2
    union always reserves the storage size for the highest storage size required by the data type of its member variables (If the union contains an array, the size is determined by the size of the highest data type or the data type of the array element multiplied by the array size, whichever is greater). In this example the data type of the member variable requiring highest storage size is int and it requires 2 bytes of storage memory.
24. a) The first member variable
25. f) a), c) and d)
26. b) False
    The preprocessor directives should not contain the character ';' as terminator
27. b) #include
28. c) #endif
29. b) #define
30. c) 9
    A and B will be replaced with 2+8 and 2+3 respectively and the expression becomes result = 2+8/2+3. The 'division operator (/) takes precedence over the addition operator and the operation becomes 2+(8/2) +3 and it will yield 2+4+3 = 9 as result.
31. a) Pointer to constant data
32. b) Constant pointer to data
33. c) Constant pointer to constant data
34. b) Pointer to volatile data
35. c) Pointer to constant volatile data

STUDY MATERIAL

36. b) Read only memory location/register
37. c) 0x80
    The instruction sets bit 7 (With index starting from 0) of flag (Initially the flag was 0x00)
38. a) Constant Data memory
39. a) 100 Bytes
    The *malloc()* memory function allocates a block of memory with specified size
40. b) realloc()

CHAPTER 10

Operating System Basics

1. e) All of these
   The kernel is the core of the operating system and is responsible for managing the system resources and the communication among the hardware and other system services. Kernel acts as the abstraction layer between system resources and user applications. Kernel contains a set of system libraries and services
2. a) System Calls
3. e) All of the above
   Process management deals with managing the processes. Process management includes setting up the memory space for the process, loading the process's code into the memory space, allocating system resources, scheduling and managing the execution of the process, setting up and managing the Process Control Block (PCB), inter process communication and synchronization, process termination/deletion etc…
4. d) All of these
5. b) Kernel space
6. d) All of these
7. d) All of these
   In monolithic kernel, all kernel services run in the kernel space. Here all kernel modules run within the same memory space under a single kernel thread. The tight internal integration of kernel modules in monolithic kernel architecture allows the effective utilization of the low-level features of the underlying system. The major drawback of monolithic kernel is that any error or failure in any one of the kernel modules will lead to the crashing of the entire kernel.
8. d) All of these
   The microkernel design incorporates only the essential set of Operating System services into the kernel. The rest of the Operating System services are implemented in programs known as '*Servers*' which runs in user space. This provides a highly modular design and OS-neutral abstraction to the kernel. Memory management, process management, timer systems and interrupt handlers are the essential services, which forms the part of the microkernel.

Introduction to Embedded Systems – Part I

Real Time Operating System (RTOS)

1. d) All of these
2. e) a) and d)
   Windows XP and Windows 2000 are desktop operating systems used for General Purpose Computing operations.
3. b) Synchronous interrupts
4. c) Divide by zero
   Almost all software interrupts are synchronous interrupts. The software interrupts are generated as a result of the instruction executed. Divide by zero software interrupt is a typical example for this.
5. d) All of these
   The time reference to RTOS kernel is provided by a high-resolution Real Time Clock (RTC) hardware chip (hardware timer). The hardware timer is programmed to interrupt the processor/controller at a fixed rate. This timer interrupt is referred as '*Timer tick*'. The '*Timer tick*' is taken as the timing reference by the kernel. The '*Timer tick*' interval may vary depending on the hardware timer. Usually the '*Timer tick*' varies in the microseconds range. The time parameters for tasks are expressed as the multiples of the '*Timer tick*'.
6. e) All of these
   Real Time Operating Systems that strictly adhere to the timing constraints for a task is referred as '*Hard Real Time*' systems. A Hard Real Time system must meet the deadlines for a task without any slippage. Missing any deadline may produce catastrophic results for Hard Real Time Systems, including permanent data lose and irrecoverable damages to the system/users. Hard Real Time Systems may not implement the virtual memory model for handling the memory. This eliminates the delay in swapping in and out the code corresponding to the task to and from the primary memory. In general, the presence of '*human in the loop (HITL)*' for tasks introduces un-expected delays in the task execution. Most of the Hard Real Time Systems are automatic and does not contain a '*human in the loop*'.
7. e) All of these

Tasks, Process and Threads

1. e) All of these
   A '*Process*' is a program, or part of it, in execution. Process is also known as an instance of a program in execution. Multiple instances of the same program can execute simultaneously. A process requires various system resources like CPU for executing the process, memory for storing the code corresponding to the process and associated variables, I/O devices for information exchange etc… A process is sequential in execution.
2. e) All of these

Introduction to Embedded Systems – Part I

A process mimics a processor in properties and holds a set of registers, process status, a Program Counter (PC) to point to the next executable instruction of the process, a stack for holding the local variables associated with the process and the code corresponding to the process.

3. a) True
4. b) Global variables
   The 'Stack' memory of a process holds all temporary data such as variables local to the process. Data memory holds all global data for the process. The code memory contains the program code (instructions) corresponding to the process.
5. a) True
6. d) 0x08FFF
   On loading a process into the main memory, a specific area of memory is allocated for the process. The stack memory starts at the highest memory address from the memory area allocated for the process.
7. b) 0x08000
   On loading a process into the main memory, a specific area of memory is allocated for the process. The code memory starts at the lowest memory address from the memory area allocated for the process.
8. c) Ready state
9. e) All of these
   When a process changes its state from Ready to running or from running to blocked or completed or from blocked to running, the CPU allocation for the process may also change. Whether the CPU allocation is to be changed is dependent on the scheduling policies adopted by the kernel for scheduling the processes.
10. d) All of these
    A thread is the primitive that can execute code. A thread is a single sequential flow of control within a process. '*Thread*' is also known as lightweight process. A process can have many threads of execution. Different threads, which are part of a process, share the same address space; meaning they share the data memory, code memory and heap memory area.
11. a) True
    Refer to the explanation given for qn 10
12. a) True
    Refer to the explanation given for qn 10
13. e) a) and b) only
    Threads of a process maintain their own thread status (CPU register values), Program Counter (PC) and stack.
14. e) All of these
15. d) All of these
16. a) Pthreads
17. c) Both of the above
    Calling a thread Object's *wait()* method in Java causes the thread object to wait. The thread will remain in the 'Wait' state until another thread

invokes the *notify()* or *notifyAll()* method of the thread object which is waiting.

18. d) All of these
    User level threads do not have kernel/Operating System support and they exist solely in the running process. Even if a process contains multiple user level threads, the OS treats it as single thread and will not switch the execution among the different threads of it. It is the responsibility of the process to schedule each thread as and when required. User level threads execute non-preemptively in relation to each other; meaning they will wait each other to get the CPU when it is used by one of them (Co-operative execution). In summary, a collection of user level threads of a process, which are serviced by the same system/kernel level thread will execute non-preemptively in relation to each other.
19. f) b), c) and d)
    Many to One, One to One and Many to Many are the different thread binding models for binding user level threads with kernel level threads.
20. a) True

Multiprocessing and Multitasking

1. a) False
   Multiprocessing is the ability to execute multiple processes simultaneously where as multitasking is the act of executing multiple processes in pseudo-parallelism.
2. b) Multiple CPUs
3. c) Multiprogramming
   Multiprogramming is an essential feature for multitasking.
4. b) Multiple process can run simultaneously
   Multiprocessing system contains multiple CPUs and can execute multiple processes simultaneously.
5. Both a) Only a single process can run at a time and c) Multiple processes run in pseudo parallelism
   Multitasking involves the execution of multiple processes in pseudo parallelism. But at any given point of time only one process is allowed to run.
6. a) CPU execution switching of processes
7. d) All of these
   Context switching forms the backbone of multitasking. Context switching involves context saving and retrieval.
8. a) Co-Operative b) Preemptive and c) Non-preemptive are the common types of multitasking present in Operating Systems supporting multitasking. Certain Operating systems doesn't support multitasking at all. They support the sequential execution of only a single task. This depends on the design of the Operating System. MSDOS is an example for such an operating system.

Introduction to Embedded Systems – Part I

9. a) The currently executing task terminates its execution & c) The currently executing task relinquishes the CPU before terminating
   In Co-operative multitasking, a process gets a chance to execute when a currently running process relinquishes the CPU. It can happen when the process terminates or when the process decides to release the CPU voluntarily before the natural termination.
10. b) The execution of a process is preempted based on the scheduling policy
    In general, preemptive multitasking preempts the execution of the currently running process based on the scheduling policy. It is not necessary that each process should get an equal chance for execution. It depends on the underlying scheduling policy adopted by the OS for preemption. The time based preemption policy like Round Robin (RR) ensures that every process gets an equal chance for execution where as the priority based preemption policy ensures that the process with highest priority always gets the highest chance for execution.
11. d) All of these
    In non-preemptive multitasking, the process/task, which is currently given the CPU time, is allowed to execute until it completes execution (enters the '*Completed*' state) or enters the '*Wait*' state or when it voluntarily relinquish the CPU.
12. a) Single user Process with single thread

Task Scheduling

1. c) scheduler
2. a) True
3. f) Any one among a) to d) depending on the type of multitasking supported by OS
   The scheduling decision is dependent on the type of multitasking supported (scheduling policy implemented) by the OS. Suppose the type of multitasking is non-preemptive, when a process changes its state from 'Wait' to 'Ready' state it is not necessary that the process is allocated the CPU time. It has to wait till the currently executing process (Process in the 'Running' state) enters either the 'Wait' state or the 'Completed' state. If the type of multitasking supported by the OS is priority based preemptive multitasking, when a process with high priority changes its state from 'Wait' to 'Ready' state and the currently executing task is a low priority task, a scheduling decision to preempt it is taken.
4. b) Preemptive
   A process changes its state to 'Ready' from 'Running' due to preemption.
5. d) b) Preemptive or c) Non-preemptive
   The processes under Preemptive and non-preemptive scheduling relinquish the CPU when they enter the wait state.
6. f) All of these
7. a) High
8. a) The number of processes executed per unit of time

Introduction to Embedded Systems – Part I

9. a) The time taken to complete its execution
10. e) All of these
11. a) Minimum
12. a) The time spent in the 'Ready queue'
13. a) Minimum
14. a) The time between the submission of a process and the first response
15. c) Least
16. f) a), c) and d)

17. b) All the processes which are 'Ready' for execution
18. d) All of these
19. c) Both of the above
    The FCFS (FIFO) scheduling policy favors CPU bound processes and I/O bound processes may have to wait a long time to complete their execution. This leads to poor device utilization.
20. a) Minimal
21. d) All of these
22. a) True
    In Shortest Job First (SJF) algorithm the 'Ready queue' is sorted based on the estimated execution completion time for a process/task. Here, the estimated execution time acts as the priority indicator. Process with shortest estimated execution completion time is given high priority. The lower the time required for completing a process the higher is its priority in SJF algorithm
23. a) A process is moved to the 'Ready' state from 'Running' state (preempted) without getting an explicit request from the process
    In preemptive scheduling the scheduler preempts a currently running task based on the priority or on time slice expiration. This operation is performed by the sole discretion of the scheduler and it doesn't involve any process intervention like process requesting for the scheduler to preempt.
24. d) b) and c)
    In SJF algorithm, a process with highest estimated execution time may not get a chance to execute at all if more and more processes with shortest estimated execution time enter the ready queue before the process with highest execution time is scheduled. In priority based scheduling, a process with lowest priority may not get a chance to execute at all if more and more process with high priority enters the 'Ready' queue before the lowest priority process is scheduled for execution. Both of these conditions may lead to 'Starvation of process.
25. c) Both of the above
    Starvation of a process happens for two reasons: Due to the non availability of CPU time for execution even if the process is 'Ready' for execution and due to the non availability of shared resources due to the exclusive use of the shared resource by other processes. The shared resource access may be controlled through mutual exclusion mechanisms

Introduction to Embedded Systems – Part I

like mutex, semaphores etc… A deadlock condition created by chained waiting for shared resources leads to starvation of processes.

26. b) Round Robin (RR) scheduling
27. b) FCFS (FIFO)
28. b) Aging
29. d) All of these
30. b) Priority based preemptive
31. b) To keep the CPU always engaged or to keep the CPU in idle mode depending on the system design
    Under desktop Operating systems where operating power is not a constraint, the IDLE TASK can be used for keeping the CPU engaged all time. Under embedded Operating Systems where operating power is a big constraint, the IDLE TASK can be used for putting the CPU into Power saving mode when there are no process 'Ready' for running.

Task Communication & Synchronization

1. a) Both of these
   Inter Process Communication (IPC) mechanism is used for sharing data between processes and for synchronizing the access of shared data between processes.
2. d) All of these
   Shared memory, messaging and Signals are the most commonly available techniques for Inter Process Communication under Windows.
3. a) Non-paged system memory
4. d) b) and c)
   Shared memory and message passing is used for sharing data between processes where as semaphores, mutex and critical section objects are used for synchronizing the shared memory access.
5. e) a) and b)
   Pipes (Named and unnamed) and memory mapped objects are examples for shared memory techniques used in windows kernel for sharing data between processes.
6. b) Passing the name of the memory mapped object
   Under Windows Operating system memory mapped object can be shared between the processes by either passing the handle of the object or by passing its name. If the handle of the memory mapped object created by a process is passed to another process for shared access, there is a possibility of closing the handle by the process which created the handle while it is in use by another process. This will throw OS level exceptions. If the name of the memory mapped object is passed for shared access among processes, processes can use this name for creating a shared memory object which will open the shared memory object already existing with the given name. The OS will maintain a usage count for the named object and it is incremented each time when a process creates/opens a memory mapped object with existing name. This will prevent the destruction of a shared

Introduction to Embedded Systems – Part I

memory object by one process while it is being accessed by another process. Hence passing the name of the memory mapped object is strongly recommended for memory mapped object based inter process communication.

7. a) Message passing is relatively free from synchronization overheads
8. a) True
9. c) SendMessage

   The *PostMessage* or *PostThreadMessage* API call is used by a thread in Windows for posting a message to its own message queue or to the message queue of another thread. The *PostMessage* API does not always guarantee the posting of messages to message queue. The *PostMessage* API will not post a message to the message queue when the message queue is full. Hence it is recommended to check the return value of *PostMessage* API to confirm the posting of message. The *SendMessage* API call sends a message to the thread specified by the handle *hWnd* and waits for the callee thread to process the message. The thread which calls the *SendMessage* API enters waiting state and waits for the message result from the thread to which the message is posted. The thread which invoked the *SendMessage* API call becomes blocked and the scheduler will not pick it up for scheduling.
10. a) Message structure
11. d) All of these

    Signals are used for asynchronous notifications where one process/thread fires a signal, indicating the occurrence of a scenario which the other process (es)/thread (s) is waiting. Signals are not queued and they do not carry any data.
12. d) All of these

    Racing or ***Race condition*** is the situation in which multiple processes compete (race) each other to access and manipulate shared data concurrently. In a Race condition the final value of the shared data depends on the process which acted on the data finally. Race condition is a result of process preemption and non-atomic accessing of shared data. If the shared data access is atomic, the shared data access operation cannot be interrupted by process preemption.
13. d) All of these

    Deadlock is the condition in which a process is waiting for a resource held by another process which is waiting for a resource held by the first process. To elaborate: Process A holds a resource 'x' and it wants a resource 'y' held by Process B. Process B is currently holding resource 'y' and it wants the resource 'x' which is currently held by Process A. Both hold the respective resources and they compete each other to get the resource held by the respective processes. The result of the competition is '*deadlock*'. None of the competing process will be able to access the resources held by other processes since they are locked by the respective processes.
14. e) All of these

Introduction to Embedded Systems – Part I

15. a) A process waits on a resource is not blocked on it and it makes frequent attempts to acquire the resource. But unable to acquire it since it is held by other process
16. a) The condition in which a high priority task needs to wait for a low priority task to release a resource which is shared between the high priority task and the low priority task.
17. c) All of these
    In priority inheritance, a low-priority task which is currently accessing (by holding the lock) a shared resource requested by a high-priority task, temporarily '*inherits*' the priority of that high-priority task, from the moment the high-priority task does the request. Boosting the priority of the low priority task to that of the priority of the task which requested the shared resource holding by the low priority task eliminates the preemption of the low priority task by other tasks whose priority are below that of the task requested the shared resource and thereby reduces the delay in waiting to get the resource requested by the high priority task. The priority of the low priority task which is temporarily boosted to high is brought to the original value when it releases the shared resource.
18. e) All of these
    In priority ceiling based priority inversion handling technique, a priority is associated with each shared resource. The priority associated to each resource is the priority of the highest priority task which uses this shared resource. This priority level is known as Ceiling priority. Whenever a task accesses a shared resource, the scheduler elevates the priority of the task to that of the ceiling priority of the resource. If the task which accesses the shared resource is a low priority task, its priority is temporarily boosted to the priority of the highest priority task to which the resource is also shared. This eliminates the pre-emption of the task by other medium priority tasks leading to priority inversion. The priority of the task is brought back to the original level once the task completes the accessing of the shared resource.
19. d) All of these
20. c) All of these
    'Critical Section' is the code memory area which holds the program instructions (piece of code) for accessing a shared resource (like shared memory, shared variables etc…). In order to synchronize the access to shared resources, the access to the critical section should be made exclusive.
21. c) Both of these
    Mutual exclusion is the technique adopted for enforcing mutually exclusive access to resources shared by processes. Mutual exclusion is very essential for preventing conditions like racing. On the other side mutual exclusion elevates the possibilities of occurring situations like deadlock in a multitasking environment.
22. c) Both of these
    Busy waiting (Spin lock) and Sleep & Wakeup are the two commonly adopted mutual exclusion policies. In Busy waiting technique, the process

is always busy in waiting in a loop checking the status of a lock which signals the availability of a resource, where as in the Sleep & Wakeup technique, the process enters sleep mode when it finds the resource requested by it is not available and the OS wakes up the process when the resource requested by the process is available.

23. d) All of these
24. e) a), b) and c)
    Mutex, semaphore, critical section and event objects are the synchronization mechanisms used for 'Sleep & Wakeup' technique based IPC synchronization.
25. e) Both a) & c)
    Only one process/thread can own the '*mutex object*' at a time. The state of a mutex object is set to signaled when it is not owned by any process/thread, and set to non-signaled when it is owned by any process/thread.
26. d) Both a) and b)
    Windows provides two synchronization wait functions namely, *WaitForSingleObject* and *WaitForMultipleObjects*. The *WaitForSingleObject* function is used for waiting for the signaling of a single object, whereas *WaitForMultipleObjects* waits for the signaling of multiple objects.
27. e) All of these
28. b) TryEnterCriticalSection
29. d) Both a) and c)
    The critical section object makes the piece of code residing inside it non-reentrant. This makes the piece of code residing inside critical section used in functions which are shared across multiple threads of a process as thread safe functions.
30. d) Both b) and c)
    Mutex objects, critical section objects and interlocked functions are used for synchronizing the access of shared variables under windows operating systems. Mutex objects can be used for synchronizing variables shared between the multiple threads of the same process or between multiple threads of different processes, where as critical section objects and interlocked functions are exclusively for accessing the variables shared between multiple threads of the same process.

CHAPTER 11

1. d) a) and c)
   VxWorks is a 'Hard Real Time', multitasking Operating System
2. c) PEND
3. c) *taskSuspend*
   The System call *taskSuspend* suspends a task from execution and changes its state to 'SUSPEND'
4. f) Only a), c) and d)

Introduction to Embedded Systems – Part I

Under VxWorks' kernel a tasks state is set to 'SUSPEND' under the following scenarios

a) The task is created with the system call *taskInit()*
b) The task is suspended with the system call *taskSuspend*
c) The execution of the task throws some exception

5. c) Memory address
   The standard VxWorks kernel architecture follows a single linear address space and all the tasks share this address space.
6. b) The scheduling becomes pre-emptive priority based with no priority resolution
   The system call *kernelTimeSlice()* sets the time slice for Round Robin scheduling. Under VxWorks kernel, Round Robin schedule is used for allocating the CPU equally to all tasks of equal priority (Resolving the priority among equal priority tasks). Calling the function *kernelTimeSlice()* with '0' as parameter disables the Round Robin scheduling.
7. c) 256
   0 to 255 with 0 being the highest.
8. f) Only a), b) and c)
   The system call *taskDelete()* terminates a task and frees the memory occupied by the task stack and task control block (TCB). The memory that is allocated by the task during its execution is not freed when the task is terminated.
9. b) *tUsrRoot*
   Under VxWorks kernel, the root task *tUsrRoot* is the first task executed by the scheduler. The function *usrRoot()* is the entry point to this task and it performs the following
10. c) Mail Slot
11. e) Only a) and b)
    Under VxWorks' kernel, when an exception occurs on executing a task, the default exception handler suspends the task that caused the error and saves the state of the task when the exception is occurred. The kernel and other tasks remain uninterrupted. If the exception is caused by an ISR, the description of the ISR is stored in a special location in the lower memory area and a system reset is initiated.
12. b) Test and Set
13. a) May lead to unacceptable Real-time response
    The scheduler locking mechanism disables task preemption and it may lead to lack of real-time response. However, the scheduler locking mechanism will not prevent the servicing of interrupt requests.
14. b) Binary Semaphore
15. b) Priority Inheritance
16. a) It runs in a separate context
    Under VxWorks' kernel, the ISRs run in a special context outside that of any task. This ensures faster interrupt response by eliminating the delay in saving the context of current task in execution when an interrupt occurs.
17. c) It depends on the processor architecture in which the kernel is running

Under VxWorks kernel, normally the ISRs share a common stack which is dedicated by the system for the ISR. If the processor architecture on which the VxWorks kernel is running, does not support separate stack for ISR, the ISR makes use of the stack of the interrupted task.

18. a) 64
    MicroC/OS-II supports 64 priority levels ranging from 0 to 63
19. a) Priority based pre-emptive scheduling
20. e) Only a) and b)
    Under MicroC/OS-II each task should have a unique priority ranging between 0 to 63. Interrupt Service Routines are not allowed to create tasks in MicroC/OS-II kernel.
21. e) Any one of these
22. a) *OSInit()*
23. c) Pipes
24. b) *OSIntEnter*
25. a) Stack of the Interrupted Task
    Under MicroC kernel, for normal processor architecture, ISRs makes use of the stack of the interrupted task for storing all CPU registers. However certain processor architecture like Motorola 68030 requires a separate stack for ISR.

CHAPTER 12

CHAPTER 13

1. a) List File
2. d) All of these
3. d) Map file
4. c) Object file
5. a) Data Record
6. b) 3
7. a) 0x0000
8. b) 02, 0C, 1F
9. a) Disassembler
10. e) a) & c)
11. c) All of these

CHAPTER 14

1. c) Room Temperature Vulcanization (RTV)
2. b) Sheet Metal

CHAPTER 15

1. c) Advertise the product

Introduction to Embedded Systems – Part I

2. a) The various phases in the life cycle of the product
3. d) All of these
4. b) Requirement Analysis phase
5. c) Design phase
6. b) Unit Testing
7. a) Linear
8. b) Documentation intensive
9. d) Spiral